**CHAPTER 1**

# INTRODUCTION

## 1.1 Overview

The term “Feed” (or news-feed in case of social networks like Facebook) is a data representation format used for providing users or subscribers with frequently updated content. Generally, a feed is populated with content that is provided by publishers that are subscribed by a user. This data can be sorted based on the time of release or through other factors that help in acquiring the importance or relevance of the data to that user.

The steps involved at generating a unique feed for every user is decided based on the content that is served at the end. Several questions need to be answered to decide the technique required such as:

- Will the feed be infinitely long or of fixed length?
- Will the contents of the feed be sorted in the chronological order or other user preferential factors?
- Whether or not ads are a part of the feed to monetize the feed service, etc.

After answering the above questions one needs to choose a mode for publishing the feed. They are primarily of two kinds:

- “Push” Model/Fan-out-on-write
- “Pull” Model/or Fan-out-on-load

This project tries to implement a simple and efficient and scalable Hybrid Feed Generation algorithm that uses a combination of the above two publishing methods.

## 1.2 History

The first Twitter prototype, developed by co-founders Dorsey and Florian Weber, was used as an internal service for employees Odeo and the full version was introduced publicly on July 15, 2006. At the beginning Twitter was considered as a combination of a social network as well as a microblogging service. Twitter was one of a kind service at the time that was capable of generating custom feeds to each user making it more unique and special to that user.

On September 6, 2006 the social network Facebook had introduced a “News Feed” feature similar to twitter’s. On 2018 Facebook announced significant changes to the News Feed, reconfiguring its algorithms to prioritize content from family members and friends over that of companies, outside media sites, or other public content. The shift is to improve the amount of meaningful content viewed.

# CHAPTER 2

# LITERATURE SURVEY

## 2.1 Activity Stream

An activity stream is a list of recent activities performed by an individual, typically on a single website[4]. For example, Facebook's News Feed is an activity stream. Activities in a stream include publishing a post, commenting on a post and ‘liking’.

A typical scenario is when a user subscribes to a topic or follows a user. When an activity is generated, all users that are subscribed are notified - either immediately or eventually when the user logs in or refreshes the feed.

Feeds can be displayed in chronological occurrence of events or tailored for the user using a custom algorithm. If multiple events are combined over time it is called an aggregated feed.

## 2.2 Models

There are two strategies for managing feed events

- Push - Each activity is pushed to a materialized feed maintained for every consumer.
- Pull - Activities are retrieved from producers when user logs in or refreshes the feed

Using the push model results in lower latency because feeds are pre-generated. This involves a huge number of writes, but reads are really fast. This can lead to higher resource consumption and potentially wasteful if users log in infrequently. In contrast, using the pull model requires processing time to generate a feed but is much more resource friendly.

Many of the popular platforms today use a combination of both. Although this leads to better performance, it significantly increases complexity of the architecture.

## 2.3 Message Queue

In modern cloud architecture, applications are decoupled into smaller, independent building blocks that are easier to develop, deploy and maintain. Message queues provide communication and coordination for these distributed applications[5]. In applications requiring a feed, each item in the activity streams generated by users need to be processed. This can be done by pushing items to a message queue while another component performs actions on these such as ranking and pushing them to other users.

There are a number of open source choices of messaging middleware systems, including Apache ActiveMQ, Apache Kafka, Apache Qpid, Beanstalkd, HTTPSQS JBoss Messaging, JORAM, RabbitMQ, Sun Open Message Queue, and Tarantool.

## 2.4 Facebook

Facebook's architecture uses a model that is primarily pull based[6]. News feeds are fetched using aggregators, which are query engines that accept user queries and retrieve items from the backend while performing aggregation, ranking and filtering. Aggregators use 'leaf' servers, which is a distributed storage layer that indexes most recent activity for each user.

## 2.5 Instagram

In contrast to Facebook, Instagram uses a predominantly push based model for generating a user's feed. When a user uploads a post, the activity is asynchronously pushed to each of the user's followers using a task manager and a message broker[7]. Cassandra, Redis, RabbitMQ, PostgreSQL and Memcached are used to manage data[8].

## 2.6 Twitter

Twitter uses a combination of both, pushing only to users that are currently active. Fanouts are implemented as microservices. Cache is critical for Twitter and protects backing stores from heavy read traffic. It uses Manhattan, Blobstore, FlockDB and Redis among others[9].

## 2.7 Yahoo

Yahoo describes architectures and techniques for large scale applications that require the generation of activity feeds in its paper “Feeding Frenzy: Selectively Materializing Users’ Event Feeds”[1]. It suggests selecting a push or pull action for each producer consumer pair based on their relative producing and querying frequencies and the cost of each action.

## 2.8 Pinterest

An implementation of the feed at Pinterest needs to be continually updated as users follow or unfollow other users and boards. New content is pushed out to the feeds of all followers. MySQL is used for persistence and HBase for backend storage[2].

**CHAPTER 3**

# OBJECTIVE OF THE PROJECT

- Develop a hybrid user specific feed fetch model to improve the performance and scalability in social applications.
- Develop a social platform application to share academic resources among students and teachers.
- Implement the hybrid feed fetch model into this social platform to demonstrate performance enhancement.

**CHAPTER 4**

# SCOPE

In this paper, we focus on addressing the following components of the feed generation model:

- Defining the feed generation problem formally as a partial view materialization problem.
- An analysis of the optimization problem that determines the events that needs to be pushed and the events that needs to be pulled. This describes the hybrid model that we propose, rather than using complete push or complete pull strategy, in order to minimize system load and maximize scalability while still providing latency guarantees.
- Algorithms for selectively pushing/pulling events to a consumer feed through decisions made locally on a per-producer/consumer basis and establishing the global performance benefit that it provides.
- An implementation of our techniques in an educational social networking platform to show that it performs well in practice.

Following are the components that are out of the scope of this project:

- Aggregation of events in the consumer feed.
- Revoking of events later from the feed.
- Changing position of the events in the consumer feed to avoid duplication (in case where multiple producers produce the same event).
- Diversity of the events in the feed.

# CHAPTER 5

# METHODOLOGY

According to the Yahoo paper, the optimal system utilisation can be achieved by making local push/pull decisions on a per producer-consumer pair basis, depending on the relative frequencies of publishing and querying. Our project suggests a less complex alternative by associating the push/pull decision with each user.

Here, individual users are classified as either push-target or pull-target based on the frequency of querying for feed. If a user queries for updates frequently, she's classified as a push-target. If a user queries relatively less, she's a pull-target. The threshold frequency to decide whether a user is push or pull target can be decided based on the average query frequency of all users and current system performance.

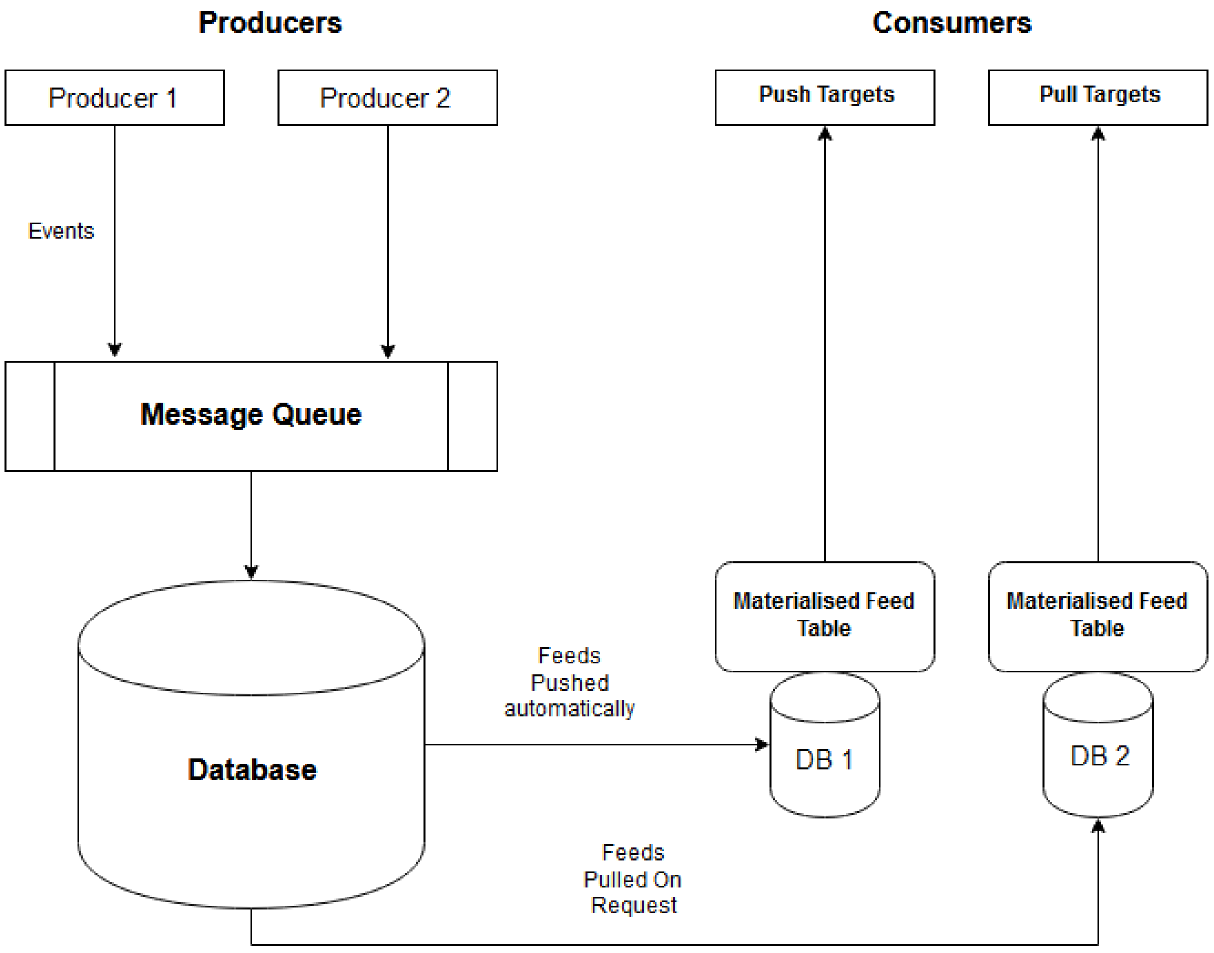

Fig 1 - Model Architecture

If a user has query frequency below the threshold, she is a pull-target; else push. When an event is generated, it is only pushed to all the followers of the producers that are push-targets. Thus when a user requests for a feed, the system first checks to see what kind of user it is. In case of a push-target, the feed is directly fetched from that user's materialized feed table. In the other case, the feed is generated by fetching the activity of the producers in that user's network. This is shown in figure 'Fig 1'.

# CHAPTER 6

# TECHNOLOGICAL REQUIREMENTS

## 6.1 Hardware Requirements

| | |
|---|---|
| Processor | Intel Pentium CPU 2.66 GHZ |
| RAM | Minimum 1GB |
| Monitor | 15” |
| Keyboard | Standard 108 Keys |

## 6.2 Software Requirements

| | |
|---|---|
| Back-end | Django |
| API | Django REST Framework |
| Front-end | React JS |
| Database | MySQL |
| Cloud Platform | Azure |

CHAPTER 7

# CONCLUSION AND FUTURE WORK

## 7.1 Conclusion

An attempt has been made to develop a hybrid scalable feed generation algorithm which tries to achieve the functionality of traditional feed generation systems and at the same time making its approach simpler and efficient. This project would give us an insight into the various ways in which different social networks or microblogging services handle feed generation.

## 7.2 Future Work

- Implementing techniques to remove seen posts on the feed to limit size after publishing them.
- Implement techniques to improve diversity of posts.
- To include a recommendation system to make the feed more personal and unique.

# REFERENCES

1. A. Silberstein et al. "Feeding Frenzy: Selectively Materializing Users' Event Feeds" *SIGMOD*, June 6–11, 2010.
2. V. Sharma et al. "Scaling Deep Social Feeds at Pinterest" *SocialCom,* 2013.
3. J. Dean. "Designs, lessons and advice from building large distributed systems" *3rd ACM SIGOPS International Workshop on Large Scale Distributed Systems and Middleware* 2009.
4. https://en.wikipedia.org/wiki/Activity_stream
5. https://aws.amazon.com/message-queue/
6. https://code.facebook.com/posts/781984911887151/serving-facebook-multifeed-efficiency-performance-gains-through-redesign/
7. https://blogs.vmware.com/vfabric/2013/04/how-instagram-feeds-work-celery-and-rabbitmq.html
8. https://engineering.instagram.com/what-powers-instagram-hundreds-of-instances-dozens-of-technologies-adf2e22da2ad
9. https://blog.twitter.com/engineering/en_us/topics/infrastructure/2017/the-infrastructure-behind-twitter-scale.html
10. http://highscalability.com/blog/2013/10/28/design-decisions-for-scaling-your-high-traffic-feeds.html
11. http://highscalability.com/blog/2013/7/8/the-architecture-twitter-uses-to-deal-with-150m-active-users.html
12. https://www.quora.com/What-are-the-best-practices-for-building-something-like-a-news-feed
13. https://fanout.io/docs/smartfeeds.html
14. https://www.slideshare.net/danmckinley/etsy-activity-feeds-architecture
15. https://stream-framework.readthedocs.io/en/latest/

Signature of Guide

Signature of Coordinator

Signature of HOD